WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

SOLUTION OF THREE RELATED COMBINATORIAL PROBLEMS BY NETWORK-FLOW AND "BRANCH AND BOUND" METHODS

Christopher R. Sprague

March 1970

451-70

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

SOLUTION OF THREE RELATED COMBINATORIAL PROBLEMS BY NETWORK-FLOW AND "BRANCH AND BOUND" METHODS

Christopher R. Sprague

March 1970

451-70

I am indebted to J.D.C. Little, D.N. Ness and G.A. Moulton for their aid.

SOLUTION OF THREE RELATED COMBINATORIAL PROBLEMS BY NETWORK-FLOW AND "BRANCH AND BOUND" METHODS

By

Christopher R. Sprague

ABSTRACT

We describe a highly efficient computational procedure for the Assignment Problem, based on Ford-Fulkerson Network-Flow methods. The Travelling Salesman Problem can be viewed as an Assignment Problem with side constraints, and "Branch and Bound" methods due to Eastman and Little can be used to extend the Assignment Problem procedure for the Travelling Salesman Problem. A more constrained problem, here called the "Travelling Gourmet Problem" can be solved by straightforward extensions to the Travelling Salesman procedure.

We present several alternative algorithms for the Travelling Salesman Problem and one for the Travelling Gourmet Problem. Computational results are included.

Preface

This is the first of two papers about methods for solving a particular class of combinatorial problems: this one examines the Assignment Problem and Eastman's subtour-elimination method (modified and extended) for the Travelling-Salesman Problem and a close relative, the Travelling Gourmet Problem; the second paper discusses the Capacitated Transportation Problem and its relationship to the special (but common) case of symmetric Travelling Salesman and Travelling Gourmet Problems. Both papers stress the synergy between Branch and Bound techniques for evoking subproblems, and Network-Flow methods for solving them.

This first paper, however, has another purpose as well. The algorithm presented for the Assignment Problem, while very close to the Standard Ford-Fulkerson algorithm, depends heavily on a set of techniques which D. C. Carroll has called "Semi-List Processing." The term is not well-defined, but refers to the use of lists of pointers, switches, indicators, etc. to introduce an appropriate amount of flexibility into data-handling, allowing faster computation. In this specific case one very significant improvement, "reorganization," is made possible only by the fact that lists of important entities are maintained at each stage of the computation. Many other operations also depend on lists. These techniques have been conventional wisdom among programmers for years and are not themselves particularly exciting. On the other hand, it is safe to say that the standard mathematical notation used to express algorithms in most journals

tends to obscure the best choices of computational methods.*

I often feel that the more elegantly an algorithm is stated, the more likely it is that the statement, straightforwardly coded, will produce a computationally inefficient procedure. This is not to say that the original designer of algorithms should present them as machine-level (or even ALGOL) code. That would be just as tragic in another way. It does mean that many insights into an algorithm can be obtained from attempting to produce a really efficient implementation; and if the original designer chooses not to seek high efficiency (and sometimes even if he does), then there is a distinct need for others to try.

---

*See, for example, D. N. Ness, "On the Computational Efficiency of Conventional Notations," Sloan School Working Paper -70.

I. INTRODUCTION

The Problems

This paper discusses the use of the Network-Flow methods of Ford and Fulkerson[1] in the solution of three related combinatorial problems: The Assignment Problem,[2] The Travelling Salesman Problem,[3] and the Travelling Gourmet Problem.[4] Each of these problems begins with an m x m cost matrix $C = \{c_{ij}\}$ and asks for an optimal (least-cost) permutation of the integers 1 . . . m, where each permutation is evaluated as follows:

> A permutation of the integers 1 . . . m consists of an ordering of those integers. In position 1 of this ordering is an integer j_1, $1 \leq j_1 \leq m$: in position 2 is a (different) integer j_2, $1 \leq j_2 \leq m$; and, in general, position i contains an integer j_i, $1 \leq j_i \leq m$. The cost of this permutation is $Z = \sum_{i=1}^{m} c_{ij_i}$.
>
> Another way of expressing the same requirement: Z is the sum of exactly m elements of $C = \{c_{ij}\}$, where exactly one element is chosen from each row of C and exactly one element is chosen from each column of C.

The Assignment Problem allows all possible permutations. There are thus m! possible solutions, of which it is required to find one with leaat cost (there may be ties).

The Travelling Salesman Problem has an additional constraint often called the tour constraint: consider each integer 1 . . . m as the number of a city, and the occurrence of j_i in position i of the permutation as designating a trip from city i to city j_i. Then the pairs of integers

(i, j_i) must form a connected tour visiting every city once and only once. For example, in a 5 x 5 problem, the permutation $p_1 = (2,5,4,1,3)$ represents the trips $(1,2)(2,5)(3,4)(4,1)(5,3)$. Reorganizing this in order of visit we have $(1,2)(2,5)(5,3)(3,4)(4,1)$ which is a connected tour visiting each city once and only once. On the other hand, the permutation $p_2 = (2,5,4,3,1)$ represents the trips $(1,2)(2,5)(3,4)(4,3)(5,1)$ which becomes two disconnected subtours: $(1,2)(2,5)(5,1)$ and $(3,4)(4,3)$. This latter permutation (p_2) is not a feasible solution to the Travelling Salesman Problem although it is a feasible solution to the Assignment Problem. There are thus only $(m-1)!$ possible solutions to a Travelling Salesman Problem.

The Travelling Gourmet Problem has a class of constraints in addition to those of the Travelling Salesman Problem which we shall call cluster constraints. For simplicity we here assume only one such constraint. We designate a set of cities K, called a cluster, and an associated number S_k. We require that no more than S_k of the cities in K be visited sequentially without an intervening visit to some city not in K. For example, let $K = (2,3,5)$ and $S_k = 2$. The permutation $p_1 = (2,5,4,1,3)$ which reorganizes to $(1,2)(2,5)(5,3)(3,4)(4,1)$ violates the cluster constraint since cities (2,5,3) are all visited in order and our S_k was 2. The permutation $p_3 = (2,4,5,3,1)$ reorganizes to $(1,2)(2,4)(4,3)(3,5)(5,1)$ and only 2 of the cities (2,3,5) are visited in order, thus satisfying the cluster constraint. While we have assumed only one such constraint, generalization to two or more is straightforward. There are less than $(m-1)!$ possible

solutions to a Travelling Gourmet Problem and, indeed, it is very easy to define a set of cluster constraints which reduce the number of possible solutions to zero. For example, try a 5 x 5 problem with K = (2,3,4,5), $S_k = 2$.

Applications

Applications for all three problems range from the fanciful to the practical. While the Assignment Problem has been suggested for matching brides to grooms,[5] it is used to match jobs to machines.[6] The Travelling Salesman Problem may have actually been used for scheduling salesmen, but more often for sequencing jobs through machines.[7] The Travelling Gourmet Problem was conceived to allow a gourmet to visit most economically all of the three-star restaurants[8] in France without eating the same regional specialties more than twice in succession. It has obvious application to the scheduling of sports teams, dispatching of trucks, performance of tasks in contaminated environments, etc.

II. SOLUTION OF THE ASSIGNMENT PROBLEM BY FORD-FULKERSON NETWORK-FLOW METHODS

Theory

The following is a discussion of a version of Ford and Fulkerson's Primal-Dual Method for the Transportation Problem,[10] especially adapted for the Assignment Problem and particularly well-arranged for automatic computer solution. We begin by noting that the Assignment Problem may be restated as follows:

Find an $m \times m$ matrix $X = \{x_{ij}\}$,

$x_{ij} = 0,1$ all i,j such that $Z = \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij}x_{ij}$

is minimized, subject to:

$$\sum_{i=1}^{m} x_{ij} = i, \text{ all } i$$

$$\sum_{i=1}^{m} x_{ij} = 1, \text{ all } j.$$

By duality, this is equivalent (except for the integer constraint on the x_{ij}'s) to:

Find vectors $U = \{u_i\}$, $V = \{v_j\}$ such that

$$W = \sum_{i=1}^{m} u_i + \sum_{j=1}^{m} v_j \text{ is maximized,}$$

subject to: $u_i + v_j \leq c_{ij}$, all i, j.

By complementary slackness, an optimal solution to both the primal and dual problems will occur only when $x_{ij} = 0$ everywhere that $u_i + v_j < c_{ij}$. Our procedure is then to form a feasible dual solution (an easy thing to do

-- see below) and see if a feasible solution to the primal can be found allowing only those x_{ij}'s where $u_i + v_j = c_{ij}$ to differ from zero. This so-called "restricted primal" takes the simple form of maximizing the flow in a network defined by those c_{ij}'s equal to $u_i + v_j$. If the restricted primal is feasible, we have an optimal solution. If not, the results of the attempt at primal feasibility are used to derive a new (higher cost) dual and the process is repeated. One pleasant consequence of this method is that the resulting x_{ij}'s inevitably are 0 or 1; another is that the value of $W = \sum_{i=1}^{m} u_i + \sum_{i=1}^{m} v_j$ rises monotonically throughout solution, of which more later.

In reaching a solution, we shall make use of eight m-vectors, as follows: $T = \{t_i\}$ and $F = \{f_j\}$ take the place of $x = \{x_{ij}\}$. Since no more than one x_{ij} will differ from zero in each row and in each column, the entire matrix X can be represented thus:

> If no x_{ij} in row i differs from zero, then $t_i = 0$. But if some x_{ij} in row i equals 1, then $t_i = j$. Similarly, if no x_{ij} in column j differs from zero, then $f_j = 0$. But if some x_{ij} in column j equals 1, then $f_j = i$.

$U = \{u_i\}$ and $V = \{v_j\}$ are the dual solution spoken of above; T,F,U,V and W represent the entire primal and dual solutions.

In order to understand the use of the other four m-vectors, a common vocabulary must be established. A row i will be said to be <u>labeled from the source</u> iff $t_i = 0$, i.e., no x_{ij} differs from zero. A column j will be said to be <u>labelable from row i</u> iff row i is labeled and $c_{ij} = u_i + v_j$.

While a column may be labelable from many different rows, it can at any time be labeled from no more than one. A row i is said to be labeled from column j iff column j is labeled and $x_{ij} = 1$, i.e., $f_j = i$.

$R = \{r_i\}$ contains the row labels: if row i is unlabeled, $r_i = 0$; if row i is labeled from the source, $r_i = -1$; and if row i is labeled from column j, $r_i = j$. $P = \{p_j\}$ contains the column labels: if column j is unlabeled, $p_j = 0$; and if column j is labeled from row i, $p_j = i$. At any point in the computation, some rows are labeled and others are not. The first n_s elements of $S = \{s_q\}$ are the indices of labeled rows. Similarly, some columns are labeled and the others are not. The first n_d elements of $D = \{d_h\}$ are the indices of labeled columns, while the last $m - n_d$ elements of D are the indices of unlabeled columns.

It was noted previously that the value of $W = \sum_{i=1}^{m} u_i + \sum_{i=1}^{m} v_j$ rises monotonically throughout the computation. We shall find it useful later to be able to suspend the computation if W becomes equal to or greater than some arbitrary value B. The algorithm described below includes this capability.

Forming an Initial Solution

We begin by finding an initial solution which consists of a complete feasible dual ($W = \sum_{i=1}^{m} u_i + \sum_{j=1}^{m} v_j$ and $c_{ij} \geq u_i + v_j$, all i,j) and a partial primal (at least one $x_{ij} = 1$):

1. For every i, set $u_i = \min_j (c_{ij})$

   set $t_i = 0$

2. For every j, set $v_j = \min_i (c_{ij} - u_i)$

   set $f_j = 0$

3. Set $W = \sum_{i=1}^{m} u_i + \sum_{j=1}^{m} v_j$

4. For every i, j where $c_{ij} = u_i + v_j$, and $t_i = 0$ and $f_j = 0$,
   set $t_i = j$ and set $f_j = i$

5. For every h, set $d_h = h$.

Set Initial Labels

We now set all columns unlabeled, and set all rows without assignments ($t_i = 0$) labeled from the source. Rows with assignments are set unlabeled. S will contain a list of labeled rows, and n_s will be its length. D contains a list of columns and, since all are unlabeled, n_d will be zero.

6. Set $n_d = 0$
   Set $n_s = 0$

7. For every i, set $r_i = 0$

8. For every i where $t_i = 0$,
   Set $r_i = -1$
   Set $n_s = n_s + 1$
   Set $s_{n_s} = i$

9. For every j, set $p_j = 0$

10. Set $q = 0$.

Feasibility Check

We now check to see whether all rows have assignments. If so, we are finished, and we return with the x_{ij}'s implicitly contained in T and F. W contains the objective function value.

11. If $n_s = 0$, return

Bogey Check

We now check to see whether $W < B$. If not, we return with an incomplete solution.

12. If $W \geq B$, return.

Primal Continuation Check

We now increment q (the index of s, set in step 10 or 37) and test whether there remains a labeled row to scan. If so, we continue, but if not, a revision of the dual is required.

13. Set $q = q + 1$

If $q > n_s$, go to step 30.

Attempt to Label Columns

Here we scan a labeled row $i = s_q$ to see whether an unlabeled column j exists for which $c_{ij} = u_i + v_j$. If so, this column can be labeled. If not, we return to step 13 for another labeled row.

14. Set $i = s_q$

Set $h = n_d$

15. Set $h = h + 1$

16. If $h > m$, go to step 13

Set $j = d_h$

If $c_{ij} \neq u_i + v_j$, go to step 15.

Label a Column

Here we label the column found in step 16. If this column already has an assignment ($f_j \neq 0$) we can label a new row (f_j) and continue searching row i for labelable columns. If not, we can make a new assignment. Step 17 labels column j, reorders d_h, and increments n_d to preserve the convention that the first n_d elements of D call out labeled columns, while the last $m - n_d$ elements refer to unlabeled columns. Note that we only search the latter part of D in this procedure.

17. Set $n_d = n_d + 1$

Set $d_h = d_{n_d}$

Set $d_{n_d} = j$

Set $p_j = i$

18. If $f_j = 0$, go to step 20.

Set a New Row Label

Here we label a new row, specifically the row in which the just-labeled column j has an assignment. This is perhaps a good point at which to mention the physical significance of labels. A row labeled from the source is one which has no assignment in it (i.e., $t_i = 0$, or $\sum_{i=1}^{m} x_{ij} = 0$) A column j labeled from row i indicates one of two things: if column j has no assignment, one could be placed in row i; if column j has an assignment $y = f_j$, it could be moved from row y to row i. A row y labeled

from column j already has an assignment in column j, but another could be made by moving the assignment in column j to row p_j.

For example, suppose row 1 is labeled from the source and column 5 is labeled from row 1. Suppose also that $f_5 = 3$, which means that row 3 is labeled from column 5. Now suppose that column 2 is labeled from row 3, but $f_2 = 0$, as in the following table:

Row or Column	t_i	r_i	f_j	p_j
1	0	-1	NA	NA
2	NA	NA	0	3
3	5	5	NA	NA
4	NA	NA	NA	NA
5	NA	NA	3	1

This means that an assignment could be placed in row 3, column 2. There would then be two assignments in row 3, but row 3 is labeled from column 5, and column 5 is labeled from row 1, so we move the assignment in row 3, column 5 to row 1, column 5. Row 1 was labeled from the source, so it had room for an assignment.

Step 19 puts a label on the row in which a labeled column has an assignment.

19. Set $y = f_j$
    Set $r_y = j$
    Set $n_s = n_s + 1$
    Set $s_{n_s} = y$
    So to step 15.

Make a New Assignment

Here we follow the procedure outlined above to make a new assignment, following the chain of labels back to the source. As we go back to the source, we set each row label in the chain to zero, to indicate that it is no longer valid.

20. Set $i = p_j$

    Set $t_i = j$

    Set $f_j = i$

    Set $j = r_i$

    Set $r_i = 0$

21. If $j \neq -1$, go to step 20.

Unwind Labels

Here we contract the lists of labels to account for those no longer valid because of a new assignment. The straightforward way of coping with this problem is to effectively discard all labels by returning to step 6, but considerable computation may have gone into producing those labels some of which would simply reappear, so some effort is called for. Basically, we have two loops. The first, steps 22-26, removes the labels from columns which have no intact chain back to the source. The second, steps 27-29A, removes the labels from rows which are labeled from unlabeled columns, and it also removes unlabeled rows from S. At the end of both, we return to step 10.

22. Set $h = 0$

23. Set $h = h + 1$

24. If $h > n_d$, go to step 27

    Set $j = d_h$

25. Set $i = p_j$

    If $i = 0$, go to setp 26

    Set $j = r_i$

    If $j = -1$, go to step 23

    If $j \neq 0$, go to step 25

26. Set $j = d_h$

    Set $d_h = d_{n_d}$

    Set $d_{n_d} = j$

    Set $p_j = 0$

    Set $n_d = n_d - 1$

    Go to step 24

27. Set $a = 0$

    Set $q = 0$

28. Set $q = q + 1$

    If $q > n_s$, go to step 29A

    Set $i = s_q$

    If $r_i = 0$, go to step 28

    If $r_i = -1$ go to step 29

    Set $j = r_i$

    If $p_j \neq 0$, go to step 29

    Set $r_i = 0$

29. Set $a = a + 1$

Set $s_a = i$

Go to step 28

29A. Set $n_s = a$

Go to step 10.

Revise Dual

Now we revise the dual solution, based on the primal solution. We came here from step 13 and at this point no more columns can be labeled from labeled rows. In order to make at least one more $c_{ij} = u_i + v_j$ in some labeled row, we must add some number to the u_i's of all labeled rows. But, to preserve the current assignments and labels, we must subtract the same number from the v_j's of all labeled columns. Our first task is to find that number, which is:

$$\min_{\substack{i,j \\ (i \text{ labeled, } j \text{ unlabeled})}} (c_{ij} - u_i - v_j)$$

We will also make a list of all rows in which this minimum value occurs. This will save us considerable computation when we restart the primal. In order to do this, we need a working m-vector $E = \{e_q\}$ and a length n_e.

30. Set $q = 0$

Set $a = \infty$

31. Set $q = q + 1$

If $q > n_s$, go to step 34

Set $h = n_d$

Set $i = s_q$

32. Set $h = h + 1$

If $h > m$, go to step 31

Set $j = d_h$

If $a < c_{ij} - u_i - v_j$, go to step 32

If $a = c_{ij} - u_i - v_j$, go to step 33

Set $n_e = 0$

Set $a = c_{ij} - u_i - v_j$

33. Set $n_e = n_e + 1$

Set $e_{n_e} = q$

Go to step 32.

Revise W

So, a is the number to be added to all u_i's (i labeled) and subtracted from all v_j's (j labeled).

34. For every q, $1 \leq q \leq n_s$

Set $i = s_q$

Set $u_i = u_i + a$

35. For every h, $1 \leq h \leq n_d$

Set $j = d_h$

Set $v_j = v_j - a$

36. Set $W = W + (n_s - n_d) * a$

Reorganize S

We have revised the dual solution, and can begin to label again. However, only those rows whose q-indices are called out in E contained

$c_{ij} - u_i - v_j = a$, and only those rows will now contain $c_{ij} - u_i - v_j = 0$. Therefore we reorganize S, placing these eligible rows at the end, setting q so that a return to step 12 will restart the primal in such a way that only these eligible rows will be scanned.

37. Set $q = n_s$

    For all g, $1 \leq g \leq n_e$

        Set $a = s_q$

        Set $s_q = s_{e_g}$

        Set $s_{e_g} = a$

        Set $q = q - 1$

38. Go to step 12.

This completes our description of a network-flow procedure for the assignment problem. Readers familiar with the standard versions of similar algorithms (such as for the transportation problem) will note some differences.

First, the major loops of the procedure are concerned with setting labels. Consequently, much care is taken to see that no label is ever discarded unless it must be discarded.

Second, search is reduced to a minimum by the use of what Carroll has termed "semi-list" processing. For example, X is compressed into T and F. The procedure maintains lists of labeled rows and columns so that the labels themselves are rarely tested. The keeping of lists also makes possible the reorganization performed in steps 37-38, which again results in a lessening of search time.

Third, as long as the current state of U, V, T, F and W is dual-feasible, the procedure can be safely (and profitably) entered at step 6.

We now describe a procedure for changing any given c_{ij} while maintaining dual feasibility.

Changing c_{ij}

We enter with i, j and a, the new value of c_{ij}.

41. If $a = c_{ij}$, return

    If $a < c_{ij}$ go to step 44

42. Set $c_{ij} = a$

    If $t_i \neq j$, return

    Set $y = j$

43. Set $t_i = 0$

    Set $f_y = 0$

    Return

44. Set $c_{ij} = a$

    If $c_{ij} - u_i - v_j \geq 0$ return

45. Set $W = W + c_{ij} - u_i - v_j$

    Set $u_i = c_{ij} - v_j$

    If $t_i = 0$, return

46. Set $y = t_i$

    Go to step 43.

Computational Experience

The following times, in seconds, were observed for 20 runs of the Assignment Problem procedure described above. Runs were made on an IBM 1130 located at the Sloan School of Management, M.I.T. Source language was FORTRAN.

Table I

PROBLEM	TYPE	SIZE	Method 0	1	2	3
1	RECT	20	4	4	4	4
2	"	20	6	5	6	5
3	"	40	27	19	28	20
4	"	40	31	21	31	22
5	"	60	89	66	94	67
6	"	60	85	59	86	59
7	"	80	175	119	174	120
8	"	80	206	149	205	149
9	"	100	255	185	251	180
10	"	100	252	179	250	167
11	EUCL	20	1	1	1	1
12	"	20	3	2	3	2
13	"	40	11	10	12	10
14	"	40	14	10	14	10
15	"	60	86	59	85	59
16	"	60	22	19	20	19
17	"	80	115	84	112	82
18	"	80	65	50	64	49
19	"	100	195	142	192	140
20	"	100	123	92	119	91

Key:

Type = RECT -- Costs were random 3-digit rectangularly distributed integers 0-999. All diagonals were set to ∞ .

Type = EUCL -- Costs were linear distances rounded to integers, between points located on a cartesian plane. Points were located randomly be choosing integer X and Y coordinates rectangularly distributed in the range 0-999. All diagonals were set to ∞ .

Method = 0 -- As described, but no unwinding (steps 22-29A) and no reorganization (steps 37-38).

Method = 1 -- As described, but no unwinding

Method = 2 -- As described, but no reorganization

Method = 3 -- As described.

We can see that reorganization appears to reduce the required time by 25% or more. Unwinding, on the other hand, seems to be of value primarily for larger problems, and is of marginal help even then. But we expect unwinding to help more for larger problems, because rebuilding labels must be an m^2 process, while unwinding itself, as can be seen from the description, should go as less than m^2.

To test the ability of the Assignment Problem procedure to re-solve after changes in C, the following was tried:

After a problem was solved, one row was chosen at random and the c_{ij} corresponding to the assignment in the chosen row was changed to ∞ .

The problem was then solved again. Then another different row was chosen, another ∞ put into it, and so on until m c_{ij}'s -- one in each row -- had been set to ∞ and the problem had been solved a total of m + 1 times.

The process was then reversed and each c_{ij} was restored to the original value, again in random order, and again with a new solution after each change, for a total of 2m + 1 solutions. Table II summarizes the results. The problems are a subset of those in Table I. T_1 is the time for the first solution and corresponds exactly to solution time by method 3 in Table I. T_2 is the mean time for each re-solution after an upward revision in a c_{ij}. T_3 is the mean time for each re-solution after a downward revision.

Table II

PROBLEM	TYPE	SIZE	T_1	T_2	T_3
1	RECT	20	4	.75	.40
2	"	20	5	.65	.40
3	"	40	20	2.33	1.85
4	"	40	22	2.05	1.15
5	"	60	67	5.17	5.00
6	"	60	59	4.73	3.42
11	EUCL	20	1	.55	.35
12	"	20	2	.65	.45
13	"	40	10	2.08	.88
14	"	40	10	2.00	1.25
15	"	60	59	3.58	2.12
16	"	60	19	3.00	1.58

There is clearly a substantial advantage to restarting the Assignment Problem procedure with a partial solution. This will prove to be of significant value in our attack on the Travelling Salesman Problem in the next section.

III. SOLUTION OF THE TRAVELLING SALESMAN PROBLEM BY VARIANTS OF EASTMAN'S SUBTOUR-ELIMINATION METHOD

Though the Travelling Salesman Problem has fewer solutions than does the equivalent Assignment Problem, it has proven much harder to solve. Of exact methods for its solution, the "Branch and Bound"[11] methods of Little et. al.,[12] Eastman[13] and Shapiro[14] seem most useful for problems of non-trivial size. We begin by describing the elements of Branch and Bound methods for the Travelling Salesman Problem.

Branch and Bound

We begin with the original problem which we mark "open." We also establish a best solution value or "bogey" equal to ∞ .

1. Choice: Choose some open problem (initially, there is only one). Mark it "closed."
2. Attempt to Solve: Take the problem chosen in step 1 and attempt to solve it "by inspection." If it is possible to do so, go to step 5. If not, go to step 3.
3. Branching: By some method, divide the problem chosen (now called the parent problem) into two or more new problems, each of which is "easier" than the parent problem, but which, among them all, contain every solution feasible in the parent. Mark all these new problems "open."
4. Bounding: Examine each of the new open problems produced in step 3 and assign to each a lower bound on the value of the best solution in cash. If any such lower bound equals or exceeds "bogey," discard the new problem immediately. When finished, go

to step 7.

5. Bogey test: If the value of the solution found in 2 is strictly less than "bogey", go to Step 6. Otherwise go to Step 7.
6. Record New Bogey: Set "bogey" to the value of the solution found in 2. Record the solution itself as "tour." Now look at all problems remaining and discard any problem whose lower bound as found in step 4 is greater than or equal to "bogey."
7. Test for completion: If there remain any open problems, return to step 1. Otherwise terminate. "Tour" is the optimum solution, and "bogey" is its value.

Obviously, Branch and Bound methods can be used for problems other than the Travelling Salesman Problem. For the present, however, we confine our attention to the Travelling Salesman Problem, and we now proceed to specialize the description above to Eastman's Method, which depends on the observation that the value of an Assignment Problem solution under a matrix C is a lower bound on the value of a Travelling Salesman Problem solution under the same matrix.

Eastman's Method

Begin with the original m x m cost matrix, with all diagonal elements set to ∞. Mark it "open." Set its lower bound to 0. Set "bogey" to ∞.

E1. Choice: Choose the open problem with lowest lower bound. Mark it "closed."

E2. Attempt to solve: Submit the chosen problem to the Assignment Problem Procedure described in Section II with B = "bogey." Set the lower bound of this problem to W. If W B, there

can be no Travelling Salesman Problem solution value < "bogey" so we go to step E7. If W < B, we examine the Assignment Problem solution. If the x_{ij}'s form a tour, we go to step E6.

E3. Branching: If the x_{ij}'s do not form a tour, they form two or more subtours (connected sequences of cities shorter than m). Choose the shortest subtour. It contains, say, q($2 \leq q \leq m/2$) non-zero x_{ij}'s. Create q new open problems each of which differs from the parent problem by having a (different) c_{ij} corresponding to a non-zero x_{ij} on the subtour set to ∞. If $c_{ij} = \infty$, x_{ij} is said to be "barred." Note that in none of these new problems can the same subtour recur (one of the c_{ij}'s is now ∞) but any tour (which must contain no more than q - 1 of these x_{ij}'s) can occur in at least one of the new problems.

E4. Bounding: Set the lower bound of each of the new problems to the lower bound of the parent problem. Go to step E7.

E5. Bogey test: This step has been absorbed into step 2.

E6. Record new bogey: Set "bogey" = W. Set "tour" = T. Discard all problems whose lower bound equals or exceeds bogey.

E7. Test for completion: If there are still one or more open problems, go to step E1. Otherwise, terminate.

This procedure requires relatively little space in a computer. One obviously needs to store the matrix C. But for each problem to be stored, one needs only space to store the lower bound, the position of the parent problems, the position of the x_{ij} to be barred, and the original value of the c_{ij} replaced by ∞.

Changing C from problem x, say, to problem y, requires finding their lowest level common ancestor, call it z. We can then move up from problem x to z, parent by parent, restoring c_{ij}'s on the way. Then we move up from y to z, parent by parent, barring x_{ij}'s.

Use of the procedure for changing c_{ij}'s described in Section II, we maintain dual feasibility, so that at no time after the first is a complete solution of the Assignment Problem required.

Shapiro found a method essentially identical to Eastman's very effective for Travelling Salesman Problems of the type here called RECT.[15] Nevertheless, there are two relatively simple extensions to Eastman's Method which are worth considering. We proceed to describe them.

Example of a Solution by Eastman's Method

i= \ j=	1	2	3	4	5
1	∞	8	4	0	5
2	8	∞	2	6	2
3	4	2	∞	7	3
4	0	6	7	∞	1
5	5	2	3	1	∞

One initial tree of solutions has only one problem (node) called "all tours," with lower bound 0 "Bogey" is .

All tours
0 open

Bogey = ∞

Step E1 -- We choose the only open problem ("ALL TOURS") and mark it closed.

Step E2 -- We use the Assignment Problem procedure and obtain:

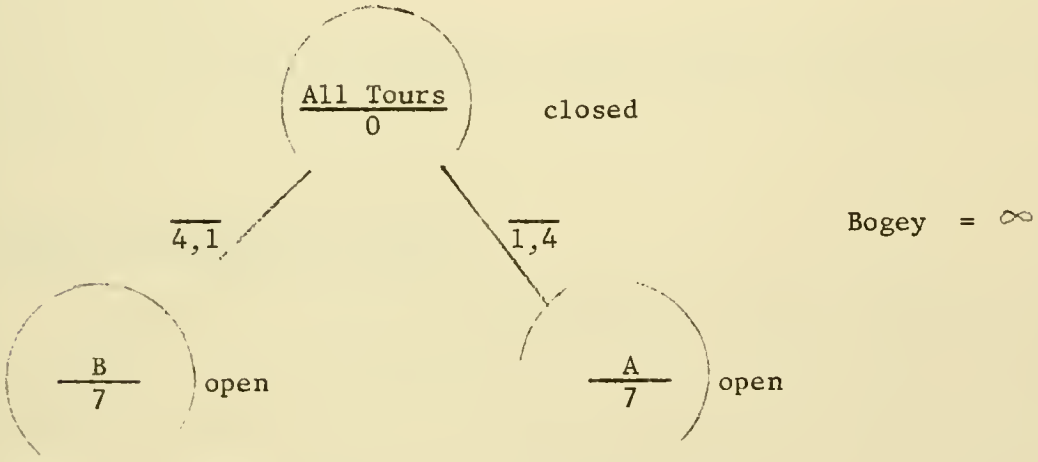
$W = 7,\ T = (4,5,2,1,3),\quad F = (4,3,5,1,2)$

$U = (2,2,3,0,3),\quad V = (0,-1,0,-2,0)$

This corresponds to two subtours, (1,4)(4,1) and (2,5)(5,3)(3,2). We go to:

Step E3 -- We create 2 new open problems, in the first, called A, we bar x_{14}; in the second, called B, we bar x_{41}.

Step E4 -- Set the lower bounds on both A and B to W which is 7. Our tree is now



Step E7 -- There are two open problems, A and B.

Step E1 -- We choose problem A and mark it closed. Using the procedure for changing one c_{ij}, we set $c_{14} = \infty$. This yields

$W = 9,\quad T = (3,5,2,1,4),\quad F = (4,3,1,5,2)$

$U = (4,2,2,0,1),\quad V = (0,0,0,0,0)$

This is a tour (1,3)(3,2)(2,5)(5,4)(4,1)

We go to:

Step E6 -- Set bogey to 9, set tour to (3,5,2,1,4).

Step E7 -- There is one open problem, B.

Step E1 -- Choose problem B and mark it closed. This requires two steps. First set the value of c_{14} to 0 (its original value). Then set the value of c_{41} to ∞ . This yields:

W = 5, T = (0,5,2,0,4), F = (0,3,0,5,2)

U = (0,2,2,0,1), V = (0,0,0,0,0)

Step E2 -- Start Assignment Problem procedure at step 6. It returns early with W = bogey = 9.

Step E7 -- There are no more open problems, Tour is (3,5,2,1,4). The cost is 9.

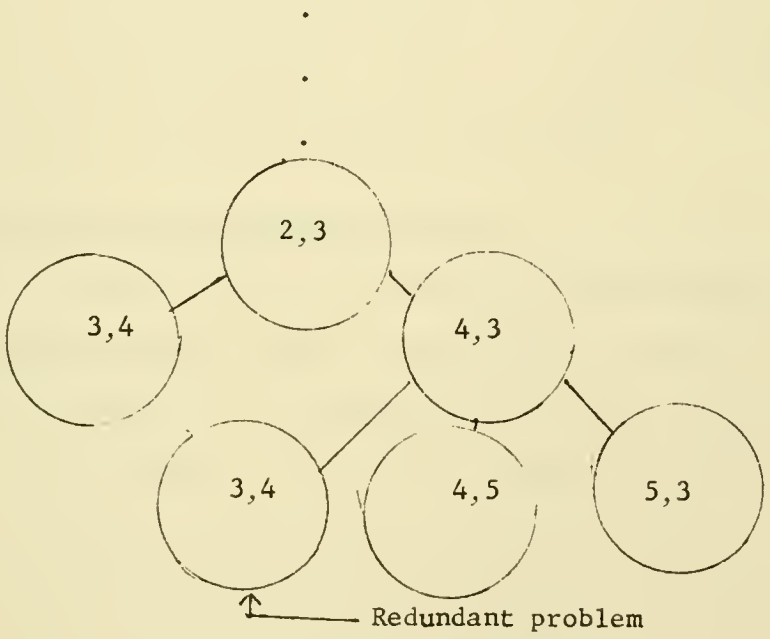
Eliminating Redundant Problems

Unfortunately, the branching procedure in Step E3 produces a set of new problems which are not mutually exclusive. In particular, any tour containing no more than q - 2 of the non-zero x_{ij}'s making up the subtour, is legal in all the new problems. We will describe Little's solution to this problem later; here we describe a method which fits within Eastman's framework.

Observe that if there be two open problems y and z such that the barred x_{ij}'s in y are a subset of the barred x_{ij}'s in z, then any tour

possible in z is also possible in y; there is no need for storing or considering z. In practice, it is difficult to find all such subset relationships, but the following procedure finds a great many of them:

As each new problem is created, we ask if any open "uncle" of the problem has the same barred x_{ij}. If so, the new problem differs from its uncle only by having one more barred x_{ij}, namely, that of its immediate parent. It is therefore redundant. If not, we try great-uncles, great-great-uncles, and so on. The diagram below shows the relationships involved. Circles represent problems, while the numbers in them represent barred x_{ij}'s.



Barring Multiple x_{ij}'s

Eastman makes the following elegant argument:

Consider a subtour of length q where $q > 2$. In order

for the tour constraint to be met, at least one city on that subtour must eventually be assigned to some city not on that subtour. Therefore, when creating a new problem corresponding to an x_{ij} on the subtour, it is permissible to bar also other x_{ij}'s corresponding to assignments of city i to other cities on the subtour. For example, consider the subtour (1,2)(2,3)(3,5)(5,1). The first new problem would normally bar x_{12}. It is also permissible to bar x_{13} and x_{15} in this new problem.

Barring a variable number of x_{ij}'s poses some problem in computation. But barring two x_{ij}'s is easy. In the above example, barring x_{12} and x_{15} assures that not only the present subtour, but also the reverse subtour, is eliminated from the new problem. This is of some help in problems of type RECT, but it is vital in problems of type EUCL.

Computational Results with Eastman's Method

A series of computer runs were made on an IBM 1620 Model II with a computer program embodying Eastman's method with elimination of redundant problems. Source language was FORTRAN except for the Assignment Problem subroutine and the redundant problem finder, both of which were hand-coded in (assembly level) SPS. All of the problems were of type RECT. The Assignment Problem routine had neither unwinding or reorganization. The program had room to store 400 problems, both open and closed. In Table III below, the symbol OF indicates a problem which overflowed core and could not be solved.

Table III

Run No.	Time in Seconds for Travelling Salesman Problem of size 20x20	40x40	60x60	80x80
1	110	135	110	795
2	30	165	310	5420
3	10	270	330	950
4	35	130	100	240
5	15	165	2655	OF
6	380	115	625	420
7	35	250	570	210
8	10	420	OF	1440
9	115	260	385	480
10	10	70	225	140
11			705	2220
12				495
13				OF
14				160
15				140
16				700
17				650
18				6060
Runs	10	10	11	18
Solved	10	10	10	16
Mean Time	75	198	602	1283
Median Time	33	165	358	573
Standard Deviation	108	98	712	1769

These results compare favorably with Shapiro's and very favorably with Little's for problems of type RECT. But there are possible alternatives to the choice, branching, and bounding schemes used in this program. Moreover, this program proved nearly worthless for problems of type EUCL.

We now proceed to investigate some alternatives in hope of improving performance.

Alternative Bounding Scheme -- Little Bounds

In step E4, we set the lower bound of each new problem to the lower bound (Assignment Problem solution value) of its parent. It is possible to develop a more powerful bound by a scheme due to Little. If, say, x_{kn} is barred, it may be possible to find new, higher values of u_k and v_n: Thus, after c_{kn} is set to ∞ :

$$u_k' = \min_j(c_{kj} - v_j) \qquad v_n' = \min_i(c_{in} - u_i)$$

These new values u_k and v_n maintain dual feasibility, and add $(u_k' + v_n' - u_k - u_n)$ to the value of W. This new value, $W + u_k' + v_u' - u_k - v_u$, is a lower bound on the value of all solutions contained in the new problem. And, of course, if this bound equals or exceeds "bogey", the new problem need not be stored at all.

Alternative Branching Scheme -- Link Inclusions

Eastman's method involves the elimination of subtours by excluding links (barring x_{ij}'s). It is possible to include links instead. (Link inclusion is central to Little's method; Eastman also suggested it). Here two new problems are created rom the parent. One of these new problems differs from its parent by barring an x_{ij}; the other is created by forcing that same x_{ij} to be non-zero ("including" a link). The latter problem must also bar another link--specifically, that x_{kn} which, should it appear in a later Assignment Problem solution, would form a subtour including the x_{ij} just included.

When branching by barring/inclusion, new problems created are mutually exclusive. Therefore, there are no redundant problems to be eliminated. Also, when solving the Assignment Problem or taking a Little bound, columns and rows corresponding to included links must be ignored. (This corresponds to Little's scheme of "crossing out" rows and columns.)

Choosing a Link for Inclusion

When branching by barring/inclusion, there is a question as to which link to bar and include. Probably one should always choose an x_{ij} which is non-zero after the Assignment Problem Solution; any other choice will simply lead to the same solution appearing again in the new problem where x_{ij} is barred. One scheme, requiring very little computation, is to follow the chain of inclusions starting at, say City 1, choosing the x_{ij} which would extend that chain one more city. We shall call this scheme "commit to chain".

Another scheme is to take Little bounds corresponding to barring each non-zero x_{ij} not already included. We then choose the x_{ij} producing the largest Little bound. We shall call this scheme "commit to best".

Alternative Choice Scheme -- Move to the Right

In Step E1, we choose the open problem with lowest lower bound (called "broad-front"). There are computational advantages to choosing an open problem very closely related to the one just solved, since the Assignment Problem procedure can be entered with as few changes as possible from the previous solution. The easy way to do this is to choose the problem most recently opened. We call this scheme "move to the right".

Normally, when branching by link inclusion, one chooses the new problem with the included link, since it is "smaller" in the sense that more rows and columns can be ignored. However, in either type of branching, one can draw a Little bound for each new problem, then choose the "best" new problem among those just opened. We call this scheme "move to best of parent". It is obviously applicable only in conjunction with "move to the right", and implies taking Little Bounds.

There are 30 meaningful combinations of all these alternatives, summarized below:

Choice and Bounding: 5 choices marked by *

0.	Choose by Broad-Front	
*0.0	Use Parent Bound	(BFPB)
*0.1	Take Little Bound	(BFLB)
1.	Move to the Right	
*1.0	Use Parent Bound	(MRPB)
*1.1	Take Little Bound	(MRLB)
*1.2	Move to Best of Parent	(MRSB)

Branching, Barring, and Redundancy: 6 choices marked by *

0.	Branch by Barring Only	
0.0	Bar One Link	
*0.0.0	Store All New Problems	(B1NR)
*0.0.1	Eliminate Redundant Problems	(B1ER)
0.1	Bar Two Links if Possible	
*0.1.0	Store All New Problems	(B2NR)
*0.1.1	Eliminate Redundant Problems	(B2ER)

1.	Branch by Inclusion	
*1.0	Commit to Chain	(CMCH)
*1.1	Commit to Best	(CMPR)

Testing Alternative Schemes

In an attempt to find out which, if any, of these alternative branching, bounding, and choice schemes could favorably affect performance, six problems were chosen to be run all 30 ways. Timings for these runs, made on an IBM 1130 at the Sloan School, are displayed in Table IV. These timings are not, unfortunately, comparable to those shown in Tables I and II, because the Assignment Problem subroutine used here is a hand-coded assembly-level version which is approximately three or four times as fast as the program timed in Tables I and II. Similarly, these timings are not comparable to those of Table III because of differences between machines.

These results indicate a strong effect of problem type, problem size, and problem itself. Beyond this, "commit to chain" is so bad as to be dismissed out of hand.

Taking Little Bounds (BFLB, MRLB) almost always decreased solution time as compared to using the parent bound (BFPB, MRPB). "Move to best of parent" (MRSB) was responsible for a dramatic decrease for one problem (No. 4), but otherwise neither helped nor hurt very much. In general, "Move to the right" was faster than "broad-front", but for one problem (again No. 4), "broad-front" was superior to all but "move to best of parent".

Eliminating redundant problems (B1ER, B2ER) was no help, except in the EUCL problems, probably because the detection of redundant problems is itself

Branching, Barring and Redundancy

		Choice and Bounding				
		BFPB	BFLB	MRPB	MRLB	MRSB
B1NR	1	27	23	26	24	36
	2	8	8	7	7	7
	3	280	156	109	85	90
	4	16	19	95	86	19
	5	205	145	50	43	43
	6	148	95	46	38	43
B1ER	1	30	25	25	24	21
	2	8	8	7	7	7
	3	260	156	95	75	70
	4	18	24	45	43	13
	5	238	177	52	46	45
	6	155	119	49	42	40
B2NR	1	28	24	26	24	37
	2	8	7	7	7	7
	3	280	160	110	90	90
	4	18	15	95	85	19
	5	131	85	40	34	36
	6	16	15	14	13	12
B2ER	1	31	26	25	25	39
	2	8	8	7	7	7
	3	275	162	100	75	70
	4	19	26	61	55	18
	5	170	100	42	37	34
	6	18	12	16	13	13
CMCH	1	280	*	120	80	210
	2	9	8	12	12	12
	3	*	*	*	*	*
	4	79	65	298	230	240
	5	**	220	104	65	70
	6	*	**	100	72	66
CMPR	1	39	35	43	34	34
	2	13	9	12	10	10
	3	130	110	90	70	68
	4	50	32	82	60	61
	5	61	40	37	28	28
	6	47	27	22	18	21

* Longer than 300 seconds
** Blew up -- i.e. exceeded core

Problems 1-2 20-city RECT
3-4 40-city RECT
5-6 10-city EUCL

Table IV

time-consuming. "Bar two if possible" (B2NR, B2ER) was of great help in the EUCL problems, because of the elimination of reverse subtours. But the clear champion branching scheme was "commit to best", (CMPR). It should be noted that this scheme differs from Little's only in that it chooses from among those x_{ij}'s set non-zero by the Assignment Problem procedure, rather than from among all $_{ij}$'s where $c_{ij} = u_i + v_j$.

In summary, the best procedure (for these six problems) was a toss-up between CMPR/MRLB and CMPR/MRSB, with B2ER a good alternative to CMPR for the EUCL problems.

More Extensive Tests of CMPR/MRSB for RECT Problems

Forty RECT problems (ten each 20, 40, 60 and 80 cities) were timed using "commit to best" and "move to best of parent". These timings are displayed in Table V. Runs were made under conditions identical with those for Table IV. (Timings for 60 and 80-city problems are to the nearest 5 seconds.)

Table V

Time in Seconds for Travelling Salesman Problems (RECT) of size

Prob	20x20	40x40	60x60	80x80
1	34	91	230	110
2	11	30	185	890
3	6	33	115	225
4	13	22	710	455
5	8	92	90	230
6	17	36	130	995
7	13	90	160	260
8	8	65	210	145
9	23	118	295	435
10	11	29	45	210
Runs	10	10	10	10
Solved	10	10	10	10
Mean Time	14	61	215	395
Median Time	12	51	180	245
Standard Deviation	8	33	175	295

IV. SOLUTION OF THE TRAVELLING GOURMET PROBLEM BY EXTENSION OF EASTMAN'S METHOD

All of the discussion in the previous section can be straightforward extended to the Travelling Gourmet Problem. All that is required is the ability to recognize the violation of a cluster constraint and to define new subproblems in which the same violation cannot occur. We then must modify two steps of the general procedure:

E1G. Choice: choose an open problem (by BFPB, BFLB, MRPB, MRLB, or MRSB). Mark it closed. If the included links (applicable to CMPR only) violate any cluster constraint, go directly to Step E7; otherwise go on to Step E2.

E3G. Branching: If the Assignment Problem Solution contains subtours, split the problem (by B1NR, B1ER, B2NR, or CMPR) and go on to Step E7.

If the Assignment Problem solution has no subtours but does violate any cluster constraint, treat the chain of links violating the constraint almost exactly as a subtour. The last S_k links in the chain tie together $S_k + 1$ cities, thus violating the cluster constraint. Split the problem into S_k new problems, each of which has one of these S_k offending links barred. Do not bar any included link. Then go to step E7.

If neither the tour constraint nor any cluster constraint is violated, go to Step E6 to record a new bogey.

The first problem chosen to be solved by this technique was an 11 - "city" problem made up of distances between 11 3-star restaurants in France. "Cities" 5,6,7, and 10 are located in Paris. "Cities" 2,8,9, and 11 are restaurants boasting predominantly Lyonnaise cuisine.

The cost matrix is shown in Table VI. Two clusters were defined: $K_1 = (5,6,7,10)$, $S_1 = 2$; and $K_2 = (2,8,9,11)$ $S_2 = 2$. The problem was first solved as a Travelling Salesman Problem using CMPR/MRSB ("commit to best", "move to best of parent"). Then the simple additions were made to activate the cluster constraints as shown above. The increase in processing time was very large, but a solution was obtained. Table VII summarizes these two runs. Conditions were the same as for Tables IV and V.

Table VI

COST MATRIX

9999	1	2	3	4	5	6	7	8	9	10	11
1	9999	436	633	673	448	448	445	405	433	448	592
2	436	9999	554	442	546	546	546	174	302	546	261
3	633	554	9999	701	234	234	234	433	461	234	346
4	673	442	701	9999	725	725	725	266	240	725	355
5	448	546	234	725	9999	0	0	459	490	0	389
6	448	546	234	725	0	9999	0	459	490	0	389
7	448	546	234	725	0	0	9999	459	490	0	389
8	405	174	433	266	459	459	459	9999	28	459	67
9	433	302	461	240	490	490	490	28	9999	490	115
10	448	546	234	725	0	0	0	459	490	9999	389
11	592	261	346	355	389	389	389	67	115	389	9999

Table VII

11 - "City" Problem Solved as:

	Travelling Salesman	Travelling Gourmet
Time in Seconds (approx)	40	550
Solution Value	2261 km	2538 km
Subproblems Evoked	1101	13190
Discarded by Little Bounds	536	5367
Discarded by Cluster Constraints	---	640
Discarded by New Bogeys	3	5
Submitted to Assignment Problem Solver	562	7178
Discarded by Assignment Problem Solver	10	352
Completed by Assignment Problem Solver	552	6826
Optimal Found in Step	128	981

V. COMMENTS ON RESULTS

First, Eastman's method, as modified and extended, is clearly very powerful for the Travelling Salesman Problem with uniform random costs (type RECT). We know of no other exact solution of problems as large as 80 x 80, nor of times so low. Perhaps more important, Little found that his technique suffered an increase in time of about 10X for every 10 extra cities. The results in Tables III and V suggest an increase of at most 2X for every 10 extra cities using Eastman's scheme.

Second, we find (like Eastman, Little and Shapiro) that symmetric, and especially Euclidean problems are very difficult: a 10-city Euclidean problem is about as hard to solve as a 40-city RECT problem. This is because of the large number of subtours of length 2 found in Assignment Problem solutions under symmetric cost matrices. Eastman suggested an extension to his method for the symmetric case. His scheme, among others, is the subject of the next paper.

Third, Branch and Bound techniques and Network-Flow techniques are both recognized and in constant use. Put together, they can be highly synergistic. This class of algorithms is a case in point.

NOTES

1. See Ford, L.R. Jr. and D.R. Folkerson, Flows in Networks, Princeton, N.J.: Princeton University Press (1962).

2. See Dantzig, G.B., Linear Programming and Extensions, Princeton, N.J.: Princeton University Press (1963), Chapter 15.

3. See Bellmore, M. and G.L. Nemhauser, "The Travelling Salesman Problem: A Survey", Operators Research 16, 538-558 (1968).

4. I am indebted to D.N. Ness for naming this problem.

5. See Dantzig

6. See Dantzig

7. See Pierce, J.F. and D. Hatfield, "Production Sequencing by Combinational Programming" in Operators Research and the Design of Management Information Systems, New York, TAPPI (1967).

8. "3-Star" as defined by Guide Michelin.

10. See Hadley, G., Linear Programming, Reading, Massachusetts: Addison-Wesley (1962) Section 10-9.

11. Little coined the term.

12. Little, J.D.C., K.G. Murty, D.W. Sweeney, and C. Karle, "An Algorithm for the Travelling Salesman Problem", Operation Research 11, 969-989 (1963).

13. Eastman, W.L. "Linear Programming with Pattern Constraints", Ph.D. Dissertation, Harvard, 1958.

14. Shapiro, D., "Algorithms for the Optimal Cost Travelling Salesman Problem", Sc.D. Thesis, Washington University, St. Louis (1966).